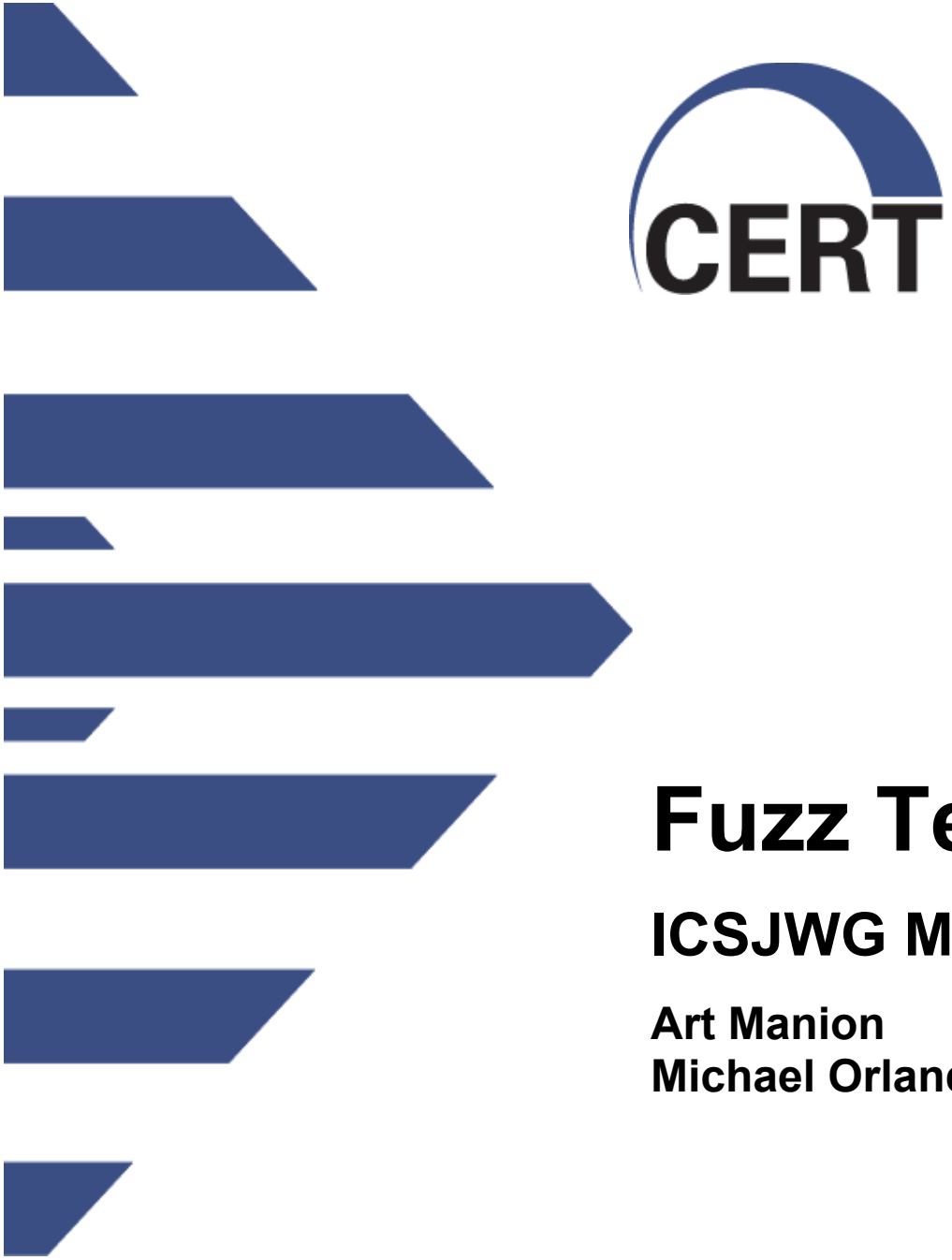


Fuzz Testing for Dummies

ICSJWG May 2011

Art Manion
Michael Orlando

Introduction

Art Manion

Michael Orlando

CERT Vulnerability Analysis team

- Analysis and research
- Coordination and disclosure
  - Vendors, researchers, other CSIRTs (including ICS-CERT)
- Discovery
  - Tools and methods to find vulnerabilities

Outline

Fuzz testing

- Tools
- ICS application

Exploit mitigation

- Microsoft Windows
  - EMET
  - ISV guidance
- UNIX-like platforms

Fuzz Testing

Providing unexpected, invalid, or random data to an application with the intention of triggering a bug

- Unexpected behavior
- Crashes
  - Buffer overflows
  - Integer overflows
  - Memory corruption
  - Format string bugs

Fuzzing Methods

Smart (generational) fuzzing

- Requires in-depth knowledge of target and specialized tools
  - Dranzer ActiveX fuzzer
- Results
  - Less crash analysis required
  - Less duplication of findings

Dumb (mutational) fuzzing

- Requires no knowledge of target, existing tools
- Results
  - More crash analysis required
  - More duplication of findings

Dumb(est) Fuzzing

Charlie Miller's "five lines of python" dumb fuzzer

- Found vulnerabilities in PDF readers and Office presentation software

```
numwrites=random.randrange(math.ceil((float(len
(buf)) / FuzzFactor)))+1for j in range
(numwrites):rbyte = random.randrange(256)rn =
random.randrange(len(buf))buf[rn] = "%c"%(rbyte);
```

<http://securityevaluators.com/files/slides/cmiller_CSW_2010.ppt>

Fuzzing Framework Requirements

Features required for an effective fuzzing framework

- Test case generation
- Application execution
- Anomaly detection
- Crash reporting

CERT Fuzzing Tools

Dranzer: Smart ActiveX fuzzer

File format fuzzers

- BFF: Basic Fuzzing Framework
- FOE: Failure Observation Engine
- Most effective against uncompressed binary formats

BFF

Debian Linux virtual machine (VMware)

- Uses zzuf, valgrind
- OS configuration optimized for fuzzing
- Software watchdog

Fuzzing scripts

- Test case generation
- Process killer
- Crash verification
- Crash de-duplication
- Crash minimization

BFF (2)

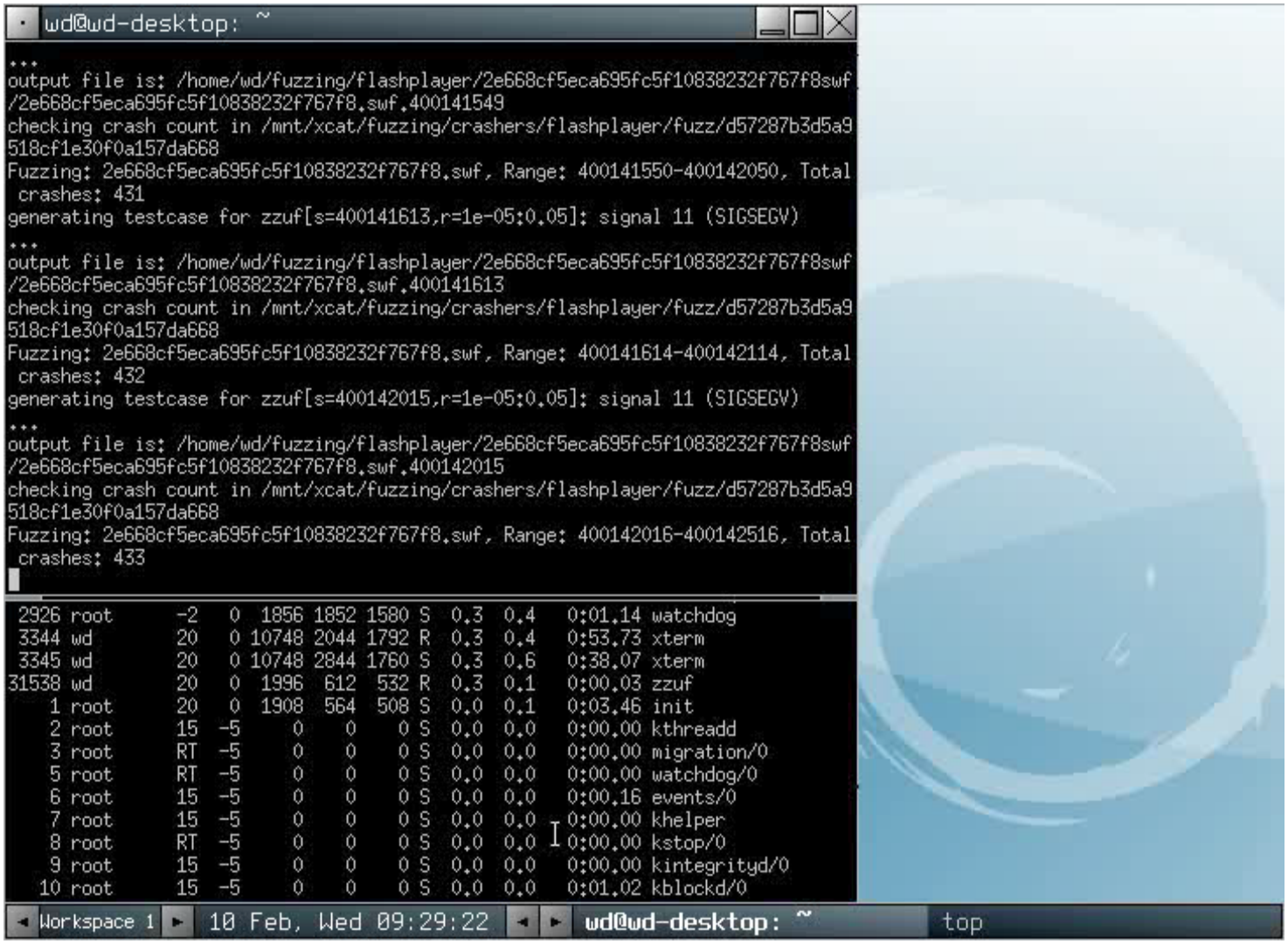
Rangefinder

- Focus on areas (bytes) in the test case that are resulting in crashes

Minimizer

- Find the least changed test case (bytes) that causes the same crash
- Inspired by fuzzdiff
- Many crashes caused by 1-3 byte changes

BFF in Action

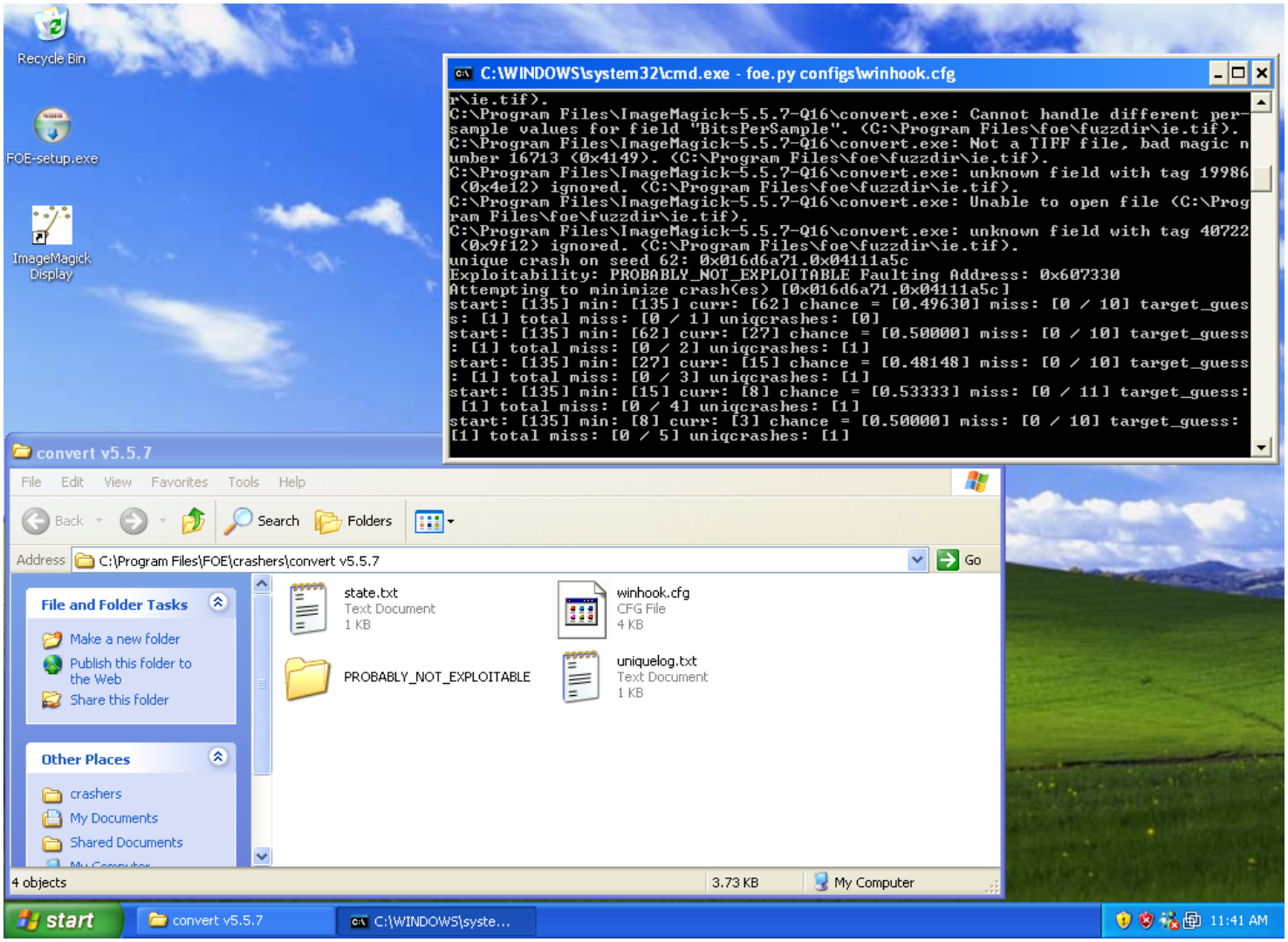


FOE

Python on Windows XP

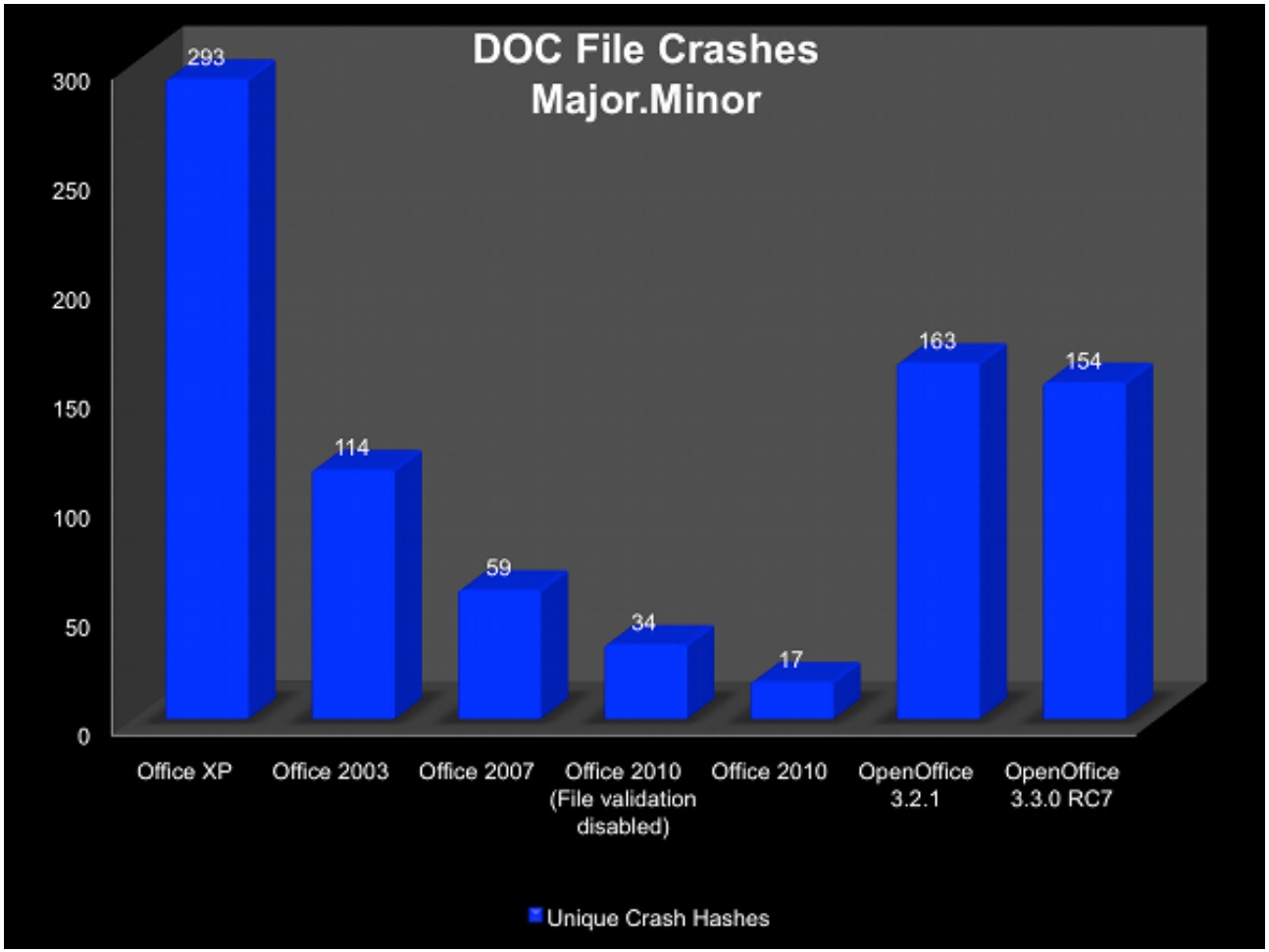
- Built from scratch
- Configurable mutators
  - bytemut, bitmut, wave, swap, copy
- Hook or full debug modes
- Output bucketing
  - Severity determination using Windows debugging extension called !exploitable (“bang exploitable”)
    - EXPLOITABLE, PROBABLY_EXPLOITABLE, PROBABLY_NOT_EXPLOITABLE, UNKNOWN

FOE in Action



Fuzzing Office Suites

A Security Comparison: Microsoft Office vs Oracle OpenOffice



Fuzzing ICS File Formats

Rockwell EDS Hardware Installation Tool (.eds)

- Previous EDS vulnerability <http://rockwellautomation.custhelp.com/app/answers/detail/a_id/67272>

Ecava IntegraXor Editor (.igx)

Automated Solutions OPC Server (.tbd)

Test Setup

Downloadable/demo software

VMware

FOE uses !exploitable to determine severity

One crash does not equal one vulnerability

250K+ iterations

Seed files

- Can affect code coverage
- Created one seed file each for .igx and .tbd
- Found ~25 .eds seed files

Results

Rockwell EDS Hardware Installation Tool (2 crashes)

- 2 EXPLOITABLE

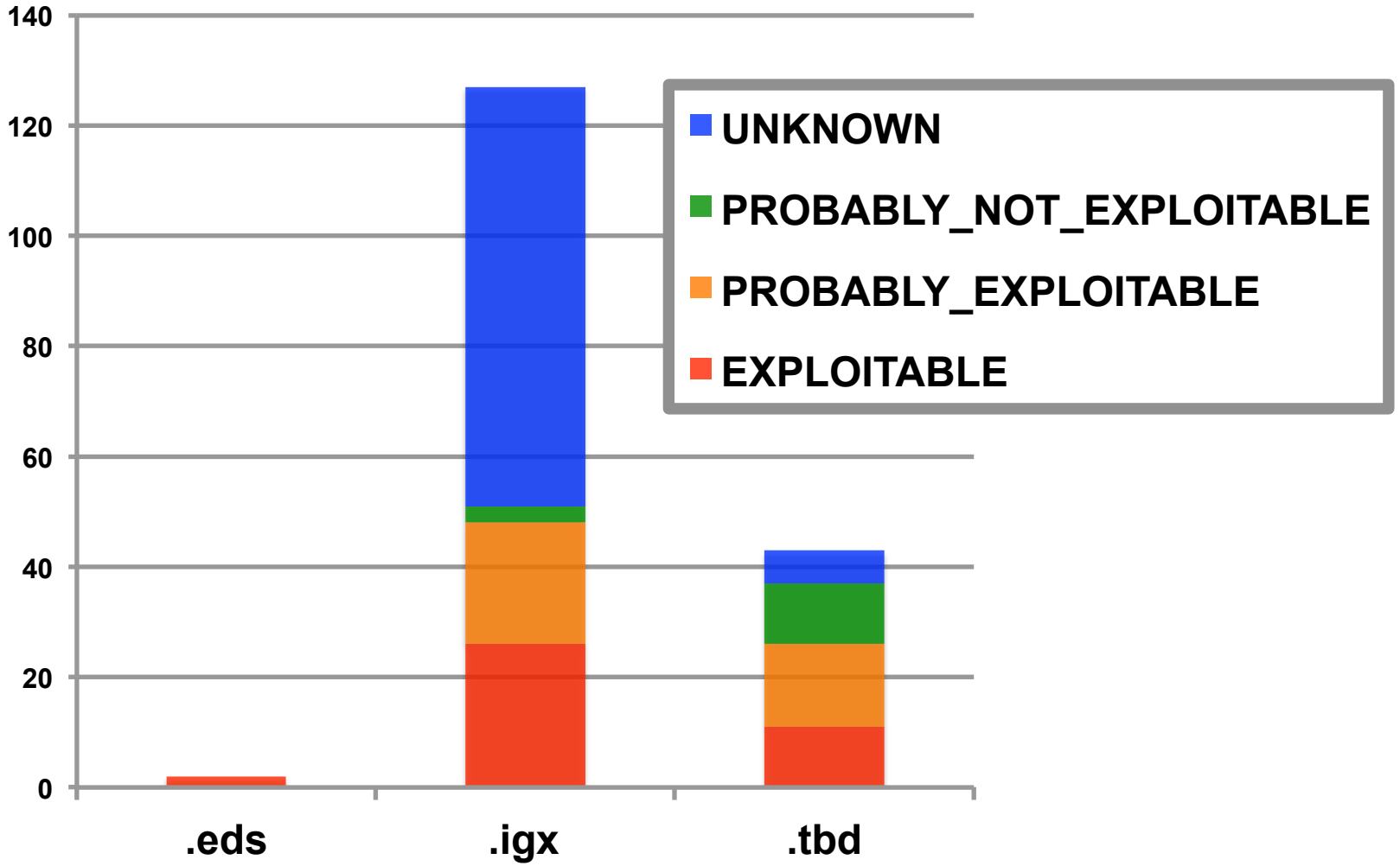
Ecava IntegraXor Editor (127 crashes)

- 26 EXPLOITABLE
- 22 PROBABLY_EXPLOITABLE
- 3 PROBABALY_NOT_EXPLOITABLE
- 76 UNKNOWN

Automated Solutions OPC Server (43 crashes)

- 11 EXPLOITABLE
- 15 PROBABLY_EXPLOITABLE
- 11 PROBABLY_NOT_EXPLOITABLE
- 6 UNKNOWN

Results (2)


140
120
100
80
60
40
20
0
.eds
.igx
.tbd
UNKNOWN
PROBABLY_NOT_EXPLOITABLE
PROBABLY_EXPLOITABLE
EXPLOITABLE

Vulnerability Mitigation

What are realistic attack vectors using ICS configuration files?

- Dangerous to load an arbitrary configuration file even in the absence of any vulnerabilities
  - ”Configuration files that are written by one user and read by another.” <http://msdn.microsoft.com/en-us/library/cc162782.aspx#Fuzzing_topic5>

Previous Rockwell Automation recommendations

- Obtain product EDS files from trusted sources (e.g. product vendor)
- Restrict physical access to the computer
- Establish policies and procedures such that only authorized individuals have administrative rights on the computer

Fuzzing Conclusions

Everything is vulnerable

- Dumb fuzzing has found vulnerabilities in everything we've targeted
- We (and others) have been focusing on common, complicated binary formats
  - PDF
  - Office document formats
  - Flash

0-day isn't rare

- Assume software you develop and run has vulnerabilities
  - You just don't know about them yet

0-Day isn't Rare

Recommendations

1. **Fuzz**
2. **Exploit mitigation**

Fuzz

Make fuzz testing part of SDLC

- No SDLC? Make dumb fuzzing the first component of your new SDLC
- CERT fuzzing tools
  - Dranzer and BFF free for download
  - FOE available by request
- Many other free and commercial tools

Somebody else is fuzzing (or is going to fuzz) your software

Vulnerabilities in Iconics GENESIS32 9.21 and GENESIS64 10.51(SCADA)
21 Mar 2011: adv1 - adv2 - adv3 - adv4 - adv5 - adv6 - adv7 - adv8 - adv9 - adv10 - adv11 - adv12 - adv13

<http://aluigi.org/adv.htm>

Exploit Mitigation: Microsoft Windows

Compile time

- Stack cookies (/GS)
- Structured Exception Handler registration (/SAFESEH)

Runtime

- Data Execution Prevention (DEP)
- Address Space Layout Randomization (ASLR)
  - Build with /DYNAMICBASE
- Heap metadata protection (HeapEnableTerminationOnCorruption)
- Structured Exception Handler Overwrite Protection (SEHOP)

Enhanced Mitigation Experience Toolkit (EMET)

Windows ISV Software Security Defenses

Exploit Mitigation: UNIX-like Platforms

Compile time

- Stack protection (StackGuard/SSP/ProPolice)
- Buffer length checking (`-D_FORTIFY_SOURCE=2 -O2`)

Runtime

- No-execute/execution disabled (NX/XD)
  - Hardware (PAE) or emulated (segment limits)
- Address randomization
  - `exec()`, `brk()`, `mmap()`, Virtual Dynamically-linked Shared Object (VDSO), Position Independent Executable (PIE)
- Global Offset Table (GOT) overwrite protection
- GNU libc heap memory manager protections
- Also Pax, W^X

Lessons Learned

Dumb fuzzing shouldn't be so effective

- Software is full of bugs, and some of those bugs are vulnerabilities
- Include fuzz testing as part of SDLC
  - Improve software security
  - Free tools from CERT and others
  - If you don't, someone else will
- Fuzzing can lead to improvements in software security

Assume everything you create and use has vulnerabilities

- Move focus from 0-day to more proactive security

Lessons Learned (2)

Exploit mitigation

- OS vendors: Implement and document exploit mitigation features
- Application vendors: Take advantage of available platform exploit mitigation features

More Information

Announcing the CERT Basic Fuzzing Framework 2.0
<http://www.cert.org/blogs/certcc/2011/02/cert_basic_fuzzing_framework_b.html>

A Security Comparison: Microsoft Office vs. Oracle OpenOffice
<http://www.cert.org/blogs/certcc/2011/04/office_shootout_microsoft_offi.html>

Automated Penetration Testing with White-Box Fuzzing
<http://msdn.microsoft.com/en-us/library/cc162782.aspx>

Windows ISV Software Security Defenses
<http://msdn.microsoft.com/en-us/library/bb430720.aspx>

The Enhanced Mitigation Experience Toolkit
<http://support.microsoft.com/kb/2458544>

Security/Features – Ubuntu Linux
<https://wiki.ubuntu.com/Security/Features>

Fuzz By Number
<http://cansecwest.com/csw08/csw08-miller.pdf>

Babysitting an Army of Monkeys
<http://securityevaluators.com/files/slides/cmiller_CSW_2010.ppt>